Functions Written

Ruomeng (Michelle) Yang

December 14, 2015

Functions & Their Uses

Here, we set up the packages, dependencies, and datasets that we need. We will also need to retrieve and read our clean dataset before we begin.

```
# Set working directory back for knitting
setwd("/Users/Michelle/Documents/UC Berkeley 2015-2016/Statistics 133/projects/final/report")

# Set up ggplot2
library(ggplot2)

# Set up readr
library(readr)

# Set up scatterplot3d
library(scatterplot3d)

# Set up stringr
library(stringr)

# Set correct working directory again
setwd("~/Documents/UC\ Berkeley\ 2015-2016/Statistics\ 133/projects/final/")

# Read data files
clean_data <- read.csv("clean_data/clean_data.csv", header = TRUE,
                       row.names = 1, stringsAsFactors = FALSE)
industries_only <- read.csv("clean_data/industries_only.csv", header = TRUE,
                            row.names = 1, stringsAsFactors = FALSE)
```

For our project, one of our tasks was to analyze which industries were in the highest and lowest beta interval. To do so, we created a variable "beta_intervals" that contained a vector sequence of 5 elements from 0 to 1.5. We then created a function that took in a beta and placed the value within the created intervals.

```
# Analyze which industries are in the highest and lowest beta interval
beta_intervals <- seq(from = 0, to = 1.5, by = 0.3)

# Places an input of beta into the correct and corresponding bin
# within beta_intervals
# input: beta (numeric) - a value specifying a certain beta
# output: correct bin (numeric) - a number 1 - 5 corresponding to the bin
interval_fun <- function(beta) {
    for (i in 1:5) {
        if (beta_intervals[i] < beta & beta < beta_intervals[i + 1]) {
            return (i)
        }
    }
```

```
}

# Place the average unlevered betas into correct bins of
# beta_intervals
beta_interval <- unlist(lapply(clean_data$Average.Unlevered.Beta,
                               FUN = interval_fun))
```

We also further applied this function to find the industries in the highest and the lowest beta intervals.

```
# Return a vector of the industries that are in a specified beta
# interval (or bin)
# input: beta_num (numeric) - the specified beta interval or bin
# output: industries (vector) - industries under the specified bin
beta_vector <- function(beta_num) {
    as.vector(clean_data$Industry[unlist(beta_interval) == beta_num])
}

# Test cases
beta_vector(5)
```

```
## [1] "Construction Supplies"             "Electronics (Consumer & Office)"
## [3] "Food Wholesalers"                  "Oilfield Svcs/Equip."
## [5] "Real Estate (General/Diversified)" "Retail (Building Supply)"
## [7] "Retail (Online)"                   "Software (Internet)"
```

```
beta_vector(1)
```

```
## [1] "Financial Svcs. (Non-bank & Insurance)"
```

Furthermore, we also wanted to know the percentage of total industries with a PEG ratio under a certain amount, so we decided to create a function to calculate this percentage.

```
# Return the percentage of total industries with a PEG ratio less than
# a specified value
# input: value (numeric) - a specified PEG ratio
# output: percent (numeric) - percent of total industries with PEG less
#                             than the input value
peg_percent_indus <- function(value) {
  peg_ratios <- clean_data$PEG.Ratio[!is.na(clean_data$PEG.Ratio)]
  length(peg_ratios[peg_ratios < value]) / length(clean_data$Industry)
}

# Testing the function
peg_percent_indus(2)
```

```
## [1] 0.8526316
```

```
peg_percent_indus(1.5)
```

```
## [1] 0.6631579
```

```
peg_percent_indus(1)
```

```
## [1] 0.2210526
```

```
peg_percent_indus(.5)
```

```
## [1] 0.03157895
```

```
peg_percent_indus(0)
```

```
## [1] 0
```

The next function returns the industry names of the values that are specified number of standard deviations above or below the mean. The input of the fuction, num_sd, is restricted to returning values that are within 0 to 4 standard deviations from the mean to keep results statistically relevant for interpretation and meaningful insight.

```
# Return the industries that have higher magnitude of expected growth
# than a specified standard deviation
# input: num_sd (numeric) - a specified standard deviation
# output: industries (vector) - industries that have an absolute
#                               growth rate above the specified
#                               standard deviation
industry_growth_outliers <- function(num_sd) {
    abs_growth <- abs(clean_data$Expected.Growth.Next.5.Years)
    mean_growth <- mean(clean_data$Expected.Growth.Next.5.Years)
    std_growth <- sd(clean_data$Expected.Growth.Next.5.Years)
    if(num_sd >= 0 & num_sd < 4) {
        return (clean_data$Industry[abs_growth >
                                        (mean_growth +
                                             num_sd * std_growth)])
    } else {
        stop(paste("Please provide standard deviations",
                   "between the values 0 and 4 as",
                   "they provide more insight."))
    }
}

# Test cases
industry_growth_outliers(1)
```

```
##  [1] "Air Transport"          "Auto & Truck"
##  [3] "Coal & Related Energy"  "Drugs (Biotechnology)"
##  [5] "Drugs (Pharmaceutical)" "Entertainment"
##  [7] "Metals & Mining"        "Oil/Gas Distribution"
##  [9] "Precious Metals"        "Retail (Online)"
## [11] "Shipbuilding & Marine"  "Software (Internet)"
```

```
industry_growth_outliers(2.2)
```

```
## [1] "Air Transport"         "Coal & Related Energy" "Shipbuilding & Marine"
```

```
# The below case is a test, which works, but must be omitted for knitting
# industry_growth_outliers(-0.83)
```

We also did quite a bit of things with removing outliers, so we decided to generalize this some more as well.

```
# Remove the outliers of two vectors and return the result as a data frame.
# input: x (vector) - input to be sorted
#        y (vector) - second input to be sorted
#        sort_x (logical) - whether to by sort x
#        sort_y (logical) - whether to by sort y
#        num_sd (int) - cutoff standard deviation for sorting
# output: removed outliers (data.frame) - two-column data frame that only
#                                         contains corresponding rows that
#                                         do not contain specified outliers
remove_outliers <- function(x, y, sort_x = TRUE, sort_y = TRUE,
                            num_sd = 2) {
    sorted_x <- sort(x, index.return = TRUE)[[2]]
    sort_x <- sort(x)
    sort_y <- y[sorted_x]
    sort_x_y <- data.frame(x = sort_x, y = sort_y)
    xm <- mean(x)
    xstd <- sd(x) * num_sd
    ym <- mean(y)
    ystd <- sd(y) * num_sd
    if (!sort_x && !sort_y) {
        stop("You have to sort by x or by y or both.")
    } else if (sort_x && sort_y) {
        dat <- subset(sort_x_y, sort_x < (xm + xstd) & sort_x > (xm - xstd) &
                          sort_y < (ym + ystd) & sort_y > (ym - ystd))
    } else if (sort_x) {
        dat <- subset(sort_x_y, sort_x < (xm + xstd) & sort_x > (xm - xstd))
    } else if (sort_y) {
        dat <- subset(sort_x_y, sort_y < (ym + ystd) & sort_y > (ym - ystd))
    }
    return (dat)
}

# Test cases
remove_outliers(clean_data$Average.Unlevered.Beta,
                clean_data$Expected.Growth.Next.5.Years, num_sd = 1)
```

```
##       x     y
## 14 0.65 14.50
## 16 0.68 20.53
## 17 0.69 11.76
## 18 0.69 11.06
## 19 0.70 10.74
## 20 0.70 11.85
## 21 0.70 10.97
## 22 0.70 15.12
## 23 0.73 11.57
```

```
## 24 0.74 16.74
## 25 0.75 10.62
## 26 0.75  8.57
## 27 0.75 13.45
## 29 0.77 14.76
## 31 0.80 14.43
## 32 0.81 15.41
## 33 0.82 14.45
## 34 0.82 20.00
## 35 0.82 13.40
## 36 0.83 13.08
## 37 0.83 10.07
## 39 0.83 16.89
## 40 0.84 19.97
## 41 0.85 14.87
## 42 0.85 19.01
## 43 0.85 15.52
## 44 0.86 17.01
## 45 0.88 10.39
## 46 0.89 18.87
## 47 0.89 17.14
## 48 0.90 13.51
## 49 0.90  8.75
## 50 0.91 13.41
## 51 0.91 15.29
## 52 0.91 12.43
## 54 0.91 14.33
## 55 0.92 16.29
## 56 0.92 13.92
## 57 0.92 12.68
## 58 0.92 18.77
## 59 0.93 18.03
## 60 0.94 12.65
## 64 0.95 14.75
## 65 0.98 11.63
## 66 0.99 10.98
## 67 0.99 14.43
## 69 0.99 16.25
## 70 1.00 18.41
## 71 1.00 16.33
## 72 1.01 16.22
## 73 1.04 14.05
## 75 1.06 10.82
## 77 1.06 20.58
## 78 1.11 15.08
## 79 1.12 11.75
```

```
# Check our test case
mean(clean_data$Average.Unlevered.Beta) +
    sd(clean_data$Average.Unlevered.Beta)
```

```
## [1] 1.126973
```

```r
mean(clean_data$Average.Unlevered.Beta) -
    sd(clean_data$Average.Unlevered.Beta)
```

```
## [1] 0.6328161
```

```r
sort(clean_data$Average.Unlevered.Beta)
```

```
##  [1] 0.06 0.34 0.37 0.41 0.42 0.43 0.51 0.53 0.58 0.59 0.59 0.59 0.61 0.65
## [15] 0.67 0.68 0.69 0.69 0.70 0.70 0.70 0.70 0.73 0.74 0.75 0.75 0.75 0.76
## [29] 0.77 0.77 0.80 0.81 0.82 0.82 0.82 0.83 0.83 0.83 0.83 0.84 0.85 0.85
## [43] 0.85 0.86 0.88 0.89 0.89 0.90 0.90 0.91 0.91 0.91 0.91 0.91 0.92 0.92
## [57] 0.92 0.92 0.93 0.94 0.94 0.94 0.95 0.95 0.98 0.99 0.99 0.99 0.99 1.00
## [71] 1.00 1.01 1.04 1.05 1.06 1.06 1.06 1.11 1.12 1.13 1.14 1.14 1.17 1.17
## [85] 1.17 1.19 1.20 1.22 1.26 1.29 1.29 1.32 1.38 1.39 1.47
```

We also wanted to generalize our graphs some more, so we made a function that both saves the graph in PNG and PDF format, if indicated, or else simply plots the graph.

```r
# Create a graph of a specified type and give users the option of
# saving the generated image in PNG and PDF format
# input: type (func) - function for high-level graphics
#        main (string) - title for plot
#        xlab (string) - label for x axis
#        ylab (string) - label for y axis
#        col (string) - color of the graph
#        ... (varies) - other input parameters for graph
#        save (logical) - whether to save graph
# output: if saved (logical) - returns if it saved
graph <- function(type, main = "", xlab = "", ylab = "",
                  col = "#000000", ..., save = FALSE) {
    if (save) {
        png(file = paste0("images/", main, ".png"))
        type(..., main = main, xlab = xlab, ylab = ylab,
             col = col)
        dev.off()

        pdf(file = paste0("images/", main, ".pdf"))
        type(..., main = main, xlab = xlab, ylab = ylab,
             col = col)
        dev.off()
    } else {
        type(..., main = main, xlab = xlab, ylab = ylab,
             col = col)
    }
    return (save)
}

# Test case - make a scatter plot and regression of
# beta to growth accounting for outliers
sorted_beta <- sort(clean_data$Average.Unlevered.Beta,
                    index.return = TRUE)[[2]]
sort_beta <- sort(clean_data$Average.Unlevered.Beta)
```

```r
sorted_growth1 <- clean_data$Expected.Growth.Next.5.Years[sorted_beta]
sorted_beta_growth <- data.frame(beta = sort_beta, growth = sorted_growth1)
dat3 <- subset(sorted_beta_growth, sorted_growth1 > 0)
graph(type = plot, dat3$growth, dat3$beta,
     col = rgb(70, 150, 80, 150, maxColorValue = 255), pch = 16,
     bg = rgb(200, 200, 200, maxColorValue = 255), save = TRUE,
     main = "function_test_scatterplot_beta_vs_growth",
     xlab = "Expected growth (%)", ylab = "Average unlevered beta")
```